Summary: This summarizes a benchmark experiment comparing Scikit-Learn and TensorFlow performance and speed on MNIST character classification via five different deep neural nets.

Research design: 55,000 of 60k samples train and 5k validate Multi-Layer Perceptron aka MLPs composed of (1) one passthrough input layer, dimensioned by flattened 28x28 = 784 pixels images + bias, (2) two or three linear threshold units, LTU, aka hidden layers, composed of 10, 25 or 50 neurons each, and (3) an output layer with a row for each of 10 digits classified. LTU activation on forward pass is by rectified linear unit aka ReLU in the hidden layers and via shared softmax function in the output layer which computes cross entropy from logits = outputs of the network. Minibatch gradient descent calculates error gradients via adaptive moment estimation aka adam optimization with learning rate set = 0.001. Neither learning rate nor regularization or dropout were explored in order to minimize treatment combinations, but early stopping was allowed, and may be a weakness to this experiment. Mean cross entropy is the loss function, but sklearn early stopping is based upon accuracy. So this design has switched TensorFlow early stopping from loss to accuracy in hopes to balance the comparison.

Python code: A 6-core 2.9 ghz MacBookPro with 32 gb 2400 mhz DDR4 runs Python v3.68, TensorFlow v1.13.1, sklearn v0.20.3 in Jupyter notebook. MNIST train and test data are separately loaded from tf.keras.datasets and assigned to their X (explanatory) and y (label) variables. X is flattened from 28x28 and standardized via sk's StandardScaler. Batch sample size and max iterations are set to 50 and max_epochs_without_progress, set to 10, stops the algorithm if accuracy has not improved for 10 iterations. Tensorflow placeholders define the input layer and tf.layers.dense define the two or three hidden layers and the output layer. An AdamOptimizer minimizes sparse_softmax_cross_entropy_with_logits mean value. A defined

function shuffles and batches the 55k train samples. 3 variables, best_loss, best_acc, and epochs_without_progress are initialized outside tf to manage early stopping. Inside the tf session, each epoch computes loss on a fed batch and a loss and accuracy on the validate data segment. Any session with a net accuracy improvement is saved off, and if more than 10 epoch's pass without further improvement, then descent is halted, the previous best is restored and test and full train set accuracy are recorded. The above processes are wrapped in Python's time.process_time()-based start and stop variables that are recorded as well. The sk algorithm is conducted in an identical fashion via the MLPClassifier model specified for the above hidden layers, batch sizes, max iterations, shuffling, validation splits, learning rates early stopping thresholds. Validation fraction is set as a percentage of the train set and though early stopping was not explicitly set = True, it was observed (possibly the older version was True always).

	Method Name	Layers	Nodes per Layer	Processing Time	Iterations	Training Set Accuracy	Test Set Accuracy
0	tf	2	10	206.962996	20	0.959417	0.9405
1	tf	2	25	383.584044	18	0.985383	0.9635
2	tf	2	50	877.591694	20	0.993900	0.9707
3	tf	3	10	249.616204	24	0.957700	0.9341
4	tf	3	25	387.691952	17	0.983250	0.9605
5	sk	2	10	474.224610	50	0.958283	0.9365
6	sk	2	25	675.659600	50	0.992350	0.9559
7	sk	2	50	809.124834	41	0.996467	0.9699
8	sk	3	10	536.258324	50	0.960017	0.9349
9	sk	3	25	633.058992	43	0.989867	0.9595

Results shown in the table above support using tensor flow over sklearn in every case for speed and generalization (comparing training with test set accuracy). TensorFlow speed is entirely due to early stopping whereas the smallest 3 of sklearn DNNs could not converge at all (those with 50 iterations). With more time, I might employ more neurons per layer. Sklearn marginally

outperforms tensorflow on 4 of 5 train accuracy measurements but in test sets, lagged tensor flow in 4 of 5 cases. Tensor flow early stopping was much earlier when triggered by loss than accuracy, but as mentioned early stopping was switched to accuracy for comparison purposes.

Management recommendations. Both models present significant flexibility that is not show in this paper. Tensorflow is tuned with many more lines of code than sklearn and so is more complex but certainly more powerful. Inasmuch as speed is a key measurement here, larger data sets and computations spread across different CPU recommend TensorFlow while Sklearn might be employed for back of the envelope computations on laptops. These results suggest NN configurations of 2 layers of 50 neurons each for MNIST data; overfitting is more a problem with 3 hidden layers.